DFT, PSD and MATLAB

Shivaraj Kandhasamy

December 14, 2010

There are three different definitions of DFT used in the literature (example see [1]),

$$\tilde{y}_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i n k}{N}} \tag{1}$$

$$\tilde{y}_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i n k}{N}} \tag{2}$$

$$\tilde{y}_k = \frac{1}{N} \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i n k}{N}} \tag{3}$$

and the corresponding inverse DFTs are defined as,

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} \tilde{y}_k e^{\frac{2\pi i n k}{N}} \tag{4}$$

$$x_n = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \tilde{y}_k e^{\frac{2\pi i n k}{N}} \tag{5}$$

$$x_n = \sum_{n=0}^{N-1} \tilde{y}_k e^{\frac{2\pi i n k}{N}} \tag{6}$$

The reason for these three definitions is that when we go from $x_n \rightarrow \tilde{y}_k \rightarrow x_n$ we need to make sure that the addition of more terms via $\sum_{n,k=0}^{N-1}$ doesn't scale the final x_n and so we need to include a factor of $\frac{1}{N}$ in the process. Where to place this factor is our choice and that's where these three definition comes into picture.

In the process of DFT (or FT), there is only one constrain that has to be satisfied by x_n and $\tilde{y}_k$ and that is the Parseval's theorem which connects the total energy in time and frequency domains (conservation of energy). Depending on which definition we use, the theorem can be expressed as follows,

$$\sum x_n^2 = \frac{1}{N}\sum \tilde{y}_k^2 \qquad \text{(DFT defined by Eq.1)} \qquad (7)$$

$$\sum x_n^2 = \sum \tilde{y}_k^2 \qquad \text{(DFT defined by Eq.2)} \qquad (8)$$

$$\sum x_n^2 = N\sum \tilde{y}_k^2 \qquad \text{(DFT defined by Eq.3)} \qquad (9)$$

As we can see DFT by using Eq. 2 doesn't need any factors of N in Parseval's theorem and seems to be a natural choice. If we define energy in time domain as x_n^2 at time t_n then total energy is $\sum x_n^2$. Similarly energy in frequency domain can be defined as $\tilde{y}_k^2$ at frequency k and total energy as $\sum \tilde{y}_k^2$. So Eq. 2 and corresponding Eq. 8 can be the guiding equations that will fix all our normalizatons (if we are interested in energy quantities). Note here that there are no factors of duration of signal T, sampling intervals dt and df. x_n and $\tilde{y}_k$ are just measurements at a particular time and particular frequency. If we want to make a correspondence between x_n and $\tilde{y}_k$ and quantities defined in continous Fourier Transform (FT) we could do so by introducing factors of T, dt and df. In principle that is not necessary, because DFT can stand on its own.

For some reason Eq. 1 (not Eq. 2) is mostly used in the DFT literature as well as in FFTs generated by computer packages like MATLAB (except MATHEMATICA which uses Eq. 2). Because of this if we use those computer packages we should be careful about factors of N and in most cases will need to rescale the $\tilde{y}_k$ such that it will match with Eq. 2.

Let us check this with a test code in MATLAB. Let us define a data set that includes three sinusoidal signals and a Gaussian random noise with $\mu = 0$ and $\sigma = 2$ (since sinusoidal signals and random noise might behave differently we need to make sure that we understand the code in both cases).

```
>> Fs = 4096; % sampling rate
>> SegmentDuration = 4;
>> N = Fs*SegmentDuration;
>> tt = 0:1/Fs:SegmentDuration-1/Fs;
>> % signal include three frequencies 100 Hz, 980 Hz, 1600 Hz.
>> x_signal = 4*sin(2*pi*100*tt)+7*sin(2*pi*980*tt)+5*sin(2*pi*1600*tt);
>> x = 2*randn(1,N)+x_signal;
>> plot(tt,x)
```

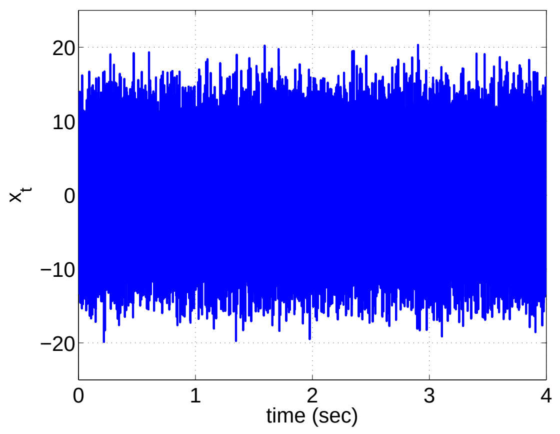

Figure 1: Time series x(t).

The resulting time series is shown in Fig. 1. Using FFT of matlab, we now calculate the (two-sided) spectrum of the above signal. Note here that MATLAB uses Eq. 1 for DFT in which signal (in frequency domain) grows as we add more and more data points. So we normalize it by N (techinically it should be $\sqrt{N}$ to comply with Eq. 8, if we are interested in the energy of the signal). The spectrum $|x_k|$ is shown in Fig. 2 (from dc to $Fs/2$).

```
>> x_k = fft(x,N)/N; % normalized by N
>> k = Fs/2*linspace(0,1,N/2+1);
>> semilogy(k,abs(x_k(1:N/2+1)))
```

One sideed power spectrum (PS) of the above signal is defined as $2|x_k|^2$ and it is shown in Fig. 3.

```
>> x_psd1 = 2*abs(x_k(1:N/2+1)).^2; %factor of 2 for one sided spectrum
>> semilogy(k,x_psd1)
```

To make sense of the numbers shown in Fig. 3, first we need to know, how signals are characterized. In general signals are classified into two categories, (i) Energy signals and (ii) Power signals. Signals which can be characterized by their total energy i.e., if their total energy is finite irrespective of the observation time, then they are called energy signals. Generally signals of transient nature fall into this category. However sinusoidal signals and random noise signals have infinite energy i.e., if we observe them long enought their total energy will go to infinite and so characterizing them by their total energy doesn't make sense. But we also know that their power (energy/sec)

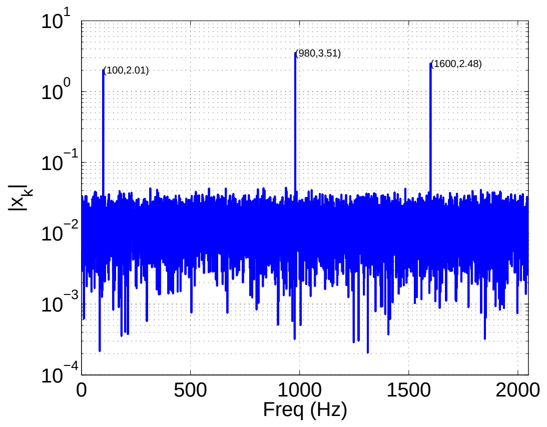

Figure 2: FFT of x(t).

is constant and is independent of the observation time and hence these kind of signals are characterized by their power (and are called power signals). It should be noted that if we limit the obseravtion time, sinusoidal or random noise signals can, in principle, be characterized by their total energy.

The definition of FFT that we are emplying here is that of Eq. 3. Even though matlab uses Eq. 1, we converted it into Eq. 3 by dividing $\tilde{y}_k$ by N (see MATLAB's technical note [2]). So all the numbers in Fig. 3 corresponds to power of the signals. A sinusoidal signal with peak value V_{peak} has power of $V^2_{\text{rms}} = \frac{V^2_{\text{peak}}}{2}$. So the power of three input sinusoidal signals are $8, 24.5, 12.5$ which indeed matches with the values shown in Fig. 3. Also the parseval theorem given by Eq. 9 holds here, both sides of that equation has a value of 8.0663×10^5 (here $\tilde{y}^2_k =$ x_psd1).

Now let us use MATLAB's in-built psd function to calculate the spectrum. The syntax for the MATLAB's psd function is

```
[psd_f,ff] = psd(x,NFFT,Fs,window,Overlaplength,detrendFlag)
```

For the first test, let us not use any windowing and overlapping. Unfortunately, if we don't use the windowing option, MATLAB chooses the default option which is a hann window ! (for the set of default values, see psdchk.m). So let us explicitly mention a rectangular window.

```
>> [x_psd2,ff] = psd(x,L,Fs,ones(L,1);
```

Here x_psd2 is not scaled properly (see psd.m header). To make any meaningful statement about power at certain frequencies we need to scale this quantity by $\frac{1}{Fs}$ and multiply by bin width $df = \frac{1}{T}$. So

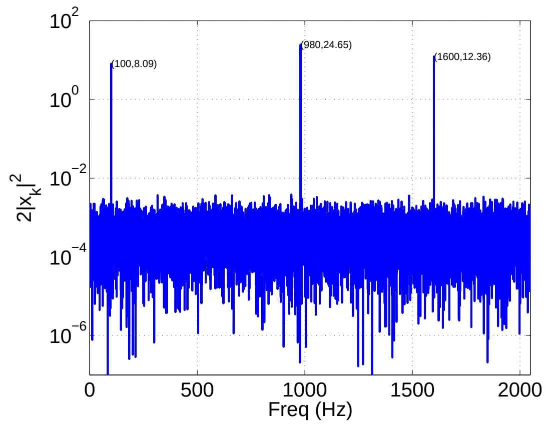

Figure 3: Power spectrum of x(t), using FFT.

```
>> x_psd2_scaled = 2*x_psd2/N; %N = Fs*SegmentDuration;
```

Fig. 4 shows the (properly scaled) power spectrum using MATLAB's in-build psd function which agrees with the one we got from using FFT of MATLAB.

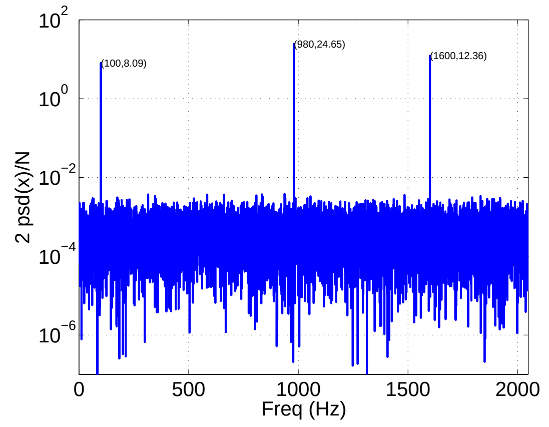

Figure 4: Power spectrum of x(t), using psd function (and scaling the results).

Note here that both Fig. 3 and Fig. 4 show power spectrum of the signal. To get the energy spectrum of the signal we need to multiply it by N (not T). Here the power is defined as energy per observation (it won't have sec^{-1} unit). In discrete signal analysis, the only factor that can come into the calcualtions is some power of N. But if we want to make connection between physical power defined as $\frac{\text{Energy}}{\text{sec}}$, then we need to introduce factors of T, df, dt (we note here that $N = \frac{T}{dt} = \frac{Fs}{df} = Fs \times T = \frac{1}{df \times dt}$).

References

[1] Heinzel G. et al., Spectrum and spectral density estimation by the Discrete Fourier transform (DFT), including a comprehensive list of window functions and some new flat-top winodws [`http://www.rssd.esa.int/SP/LISAPATHFINDER/docs/Data_Analysis/GH_FFT.pdf`]

[2] `http://www.mathworks.com/support/tech-notes/1700/1702.html`

[3] `http://www.mathworks.com/products/signal/demos.html?file=/products/demos/shipping/signal/deterministicsignalpower.html`